Designing dispatching rules with genetic programming for the unrelated machines environment with constraints

Kristijan Jaklinović[b], Marko Đurasević[a,*], Domagoj Jakobović[a]

kristijan.jaklinovic@fer.hr, marko.durasevic@fer.hr, domagoj.jakobovic@fer.hr

[a]*University of Zagreb, Faculty of Electrical Engineering and Computing, Croatia*
[b]*Infinum, Zagreb, Croatia*

Abstract

Scheduling problems constitute an important part in many everyday systems, where a variety of constraints have to be met to ensure the feasibility of schedules. These problems are often dynamic, meaning that changes occur during the execution of the system. In such cases, the methods of choice are dispatching rules (DRs), simple methods that construct the schedule by determining the next decision which needs to be performed. Designing DRs for every possible problem variant is unfeasible. Therefore, the attention has shifted towards automatic generation of DRs using different methods, most notably genetic programming (GP), which demonstrated its superiority over manually designed rules. Since many real world applications of scheduling problems include various constraints, it is required to create high quality DRs even when different constraints are considered. However, most studies focused on problems without additional constraints or only considered them briefly. The goal of this study is to examine the potential of GP to construct DRs for problems with constraints. This is achieved primarily by adapting the schedule generation scheme used in automatically designed DRs. Also, to provide GP with a better overview of the problem, a set of supplementary terminal nodes are proposed. The results show that automatically generated DRs obtain better performance than several manually designed DRs adapted for problems with constraints. Using additional terminals resulted in the construction of better DRs for some constraints, which shows that their usefulness depends on the considered constraint type. Therefore, automatically generating DRs for problems with constraints presents a better alternative than adapting existing manual DRs. This finding is important as it shows the capability of GP to construct high quality DRs for more complicated problems, which is useful for real world situations where a number of constraints can be present.

Keywords:

[*]Corresponding author

Scheduling, Unrelated machines environment, Constraints, Dispatching rules, Genetic programming, Apparent tardiness cost

1. Introduction

Scheduling is a decision process which is concerned with the allocation of certain activities or jobs onto a limited set of available resources or machines (Pinedo, 2012). Because of their complexity and presence in many real world situations, scheduling problems have been extensively researched in the literature. Some real world examples of scheduling problems include various manufacturing systems (Kofler et al., 2009; Ouelhadj and Petrovic, 2009), scheduling airplanes on runways (Cheng et al., 1999; Hansen, 2004), scheduling in railway traffic (Corman and Quaglietta, 2015), nurse rostering (Burke et al., 2004), scheduling patients for treatments (Petrovic and Castro, 2011), university timetabling (Lewis et al., 2007), and many other. Most scheduling problems of interest are NP hard, meaning that the optimal solution cannot be obtained in a reasonable amount of time. Therefore, the most prevalent way of solving such problems is by using various heuristic methods. Depending on how these heuristic methods solve the scheduling problem they are divided into *improvement* and *constructive* heuristics.

The idea of improvement heuristics is that they work with one or more, usually randomly generated, solutions (schedules) which are iteratively improved. This is done by using different operators which introduce changes in the solutions. This group of heuristics contains a wide range of metaheuristic methods (Hart et al., 2005), like genetic algorithms (GAs) (Vlašić et al., 2019, 2020), simulated annealing (Kim et al., 2002), tabu search (Lee et al., 2013), ant colony optimisation (Behnamian et al., 2009), and many others. Although metaheuristics can obtain high quality results for various scheduling problems, they usually come with a serious restriction. Namely, such methods can only be used when all the information about the problem is available, which means that scheduling is performed under static conditions. However, in many situations the information about the problem is not available from the start, but rather the jobs that need to be executed become available during the execution of the system, which means that the problem must be solved under dynamic conditions. Consequently, the method applied for solving problems under dynamic conditions needs to work for problems where all information is not available from the start, or can change during the execution of the system.

Constructive heuristics are the more appropriate choice for solving scheduling problems under dynamic conditions. Instead of searching through the entire search space of solutions, these heuristics iteratively construct the solution. Most commonly, constructive heuristics appear in the form of dispatching rules (DRs) (Maheswaran et al., 1999; Braun et al., 2001; Đurasević and Jakobović, 2018). The idea behind DRs is to decide which job to schedule on which machine only at those moments when it is absolutely required. For example, such a situation would arise when a machine becomes free and the next job needs to be scheduled on it. The way in which the job is selected is by calculating a certain priority for each job and selecting the one with the highest priority value. Because of that, DRs are adaptable to changes that can happen during the execution of the system, like the arrival of new jobs over time. Therefore, they are the most obvious choice for solving scheduling problems under dynamic conditions.

DRs, however, have certain drawbacks. Most importantly, they are difficult to design manually and that process requires expert knowledge. Since there is a plethora of different scheduling problems, one would have to design DRs for each variant, which is not feasible. For that reason a large number of studies focused on the problem of automatic design of new DRs (Branke et al., 2016; Nguyen et al., 2017). Most commonly, genetic programming (GP) (Koza, 1990; Poli et al., 2008) has been used to design new DRs automatically. The research in this area demonstrated a great potential of using GP to automatically design new DRs for scheduling problems with different criteria or conditions.

In most studies the considered scheduling problems did not include additional constraints that can be present in reality. For example, some jobs may not be executed before other jobs are completed, or certain machines might be under maintenance during fixed periods of time. As a result, DRs need to take such additional constraints into account. However, as previously denoted, such problems were rarely considered, and if they were, usually only a single constraint was included. Therefore, until now most manually and automatically designed DRs did not take into account additional constraints. Although the research on unconstrained problems is important and leads to new findings, it is also necessary to validate the methods on more difficult problems which include various constraints. Therefore, the goal of this paper is to study the potential of automatically designing new DRs for the unrelated scheduling problem with additional constraints and evaluating their effectiveness compared to several selected manually designed DRs adapted for the same constraints. The constraints that are considered in this paper are sequence dependent setup times, machine unavailability periods, machine eligibility constraints, and precedence constraints. As it is quite unlikely that in real world situations only a single constraint is present, the performance of automatically designed DRs is also tested on all the combinations of the aforementioned constraints. In that way it is possible to determine how the performance of automatically designed DRs is affected by the number and existence of constraints. The contributions of this paper can be summarised as:

1. Adaptation of automatically designed DRs for different scheduling constraints
2. Proposition and evaluation of additional terminal nodes for the considered constraints
3. Performance evaluation of automatically designed DRs in comparison to manually designed DRs for problems with single and multiple constraints.

The paper is organised as follows. Section 2 provides an overview of the literature dealing with automatic design of DRs. The unrelated machines scheduling problem is described in Section 3. Section 4 describes the methods for solving the unrelated machines environment problem with additional constraints, as well as how GP can be applied to design new DRs automatically. Section 5 describes the benchmarks and experimental design. The results obtained by the tested methods are presented in Section 6. Section 7 provides a short discussion about the obtained results, while Section 8 concludes the paper and outlines directions for possible future research.

2. Literature overview

Over the years GP became one of the most popular hyper-heuristic methods. Hyper-heuristics are methods that can be used to design novel heuristics for different kinds of problems (Burke et al., 2007, 2009, 2013). GP has been used for designing new heuristics for a variety of different optimisation problems like the knapsack and bin packing problem (Burke et al., 2012), nurse rostering (Pillay and Qu, 2018), and capacitated arc routing problem (Mei and Zhang, 2018; Liu et al., 2019).

GP was first used to evolve new DRs for the single machine (Dimopoulos and Zalzala, 1999, 2001) and job shop environments (Miyashita, 2000). These studies laid out the foundations for automatic design of DRs which would be enhanced in future works. Subsequent research extended the application of GP to other machine environments like the flexible job shop environment (Tay and Ho, 2007), parallel machines environment (Jakobović et al., 2007), and the unrelated machines environment (Đurasević et al., 2016; Đurasević and Jakobović, 2020). Aside from these standard scheduling problems, GP was successfully applied on other scheduling problem types like the order acceptance and scheduling (OAS) problem (Nguyen et al., 2013, 2014) or the resource constrained project scheduling problem (RCPSP) (Chand et al., 2018; Đumić et al., 2018). These studies demonstrated that GP can create DRs for a wide range of different scheduling problem types, outlining its versatility. Another line of research was directed towards evolving DRs that construct schedules which optimise several criteria simultaneously (Nguyen et al., 2013, 2015; Karunakaran et al., 2016; Durasević and Jakobović, 2018). It was also demonstrated that no manually designed DRs could compete with such heuristics when considering several criteria simultaneously. Since in real world problems usually a number of criteria are optimised simultaneously, this line of research proved that the performance of automatically designed DRs does not degrade in cases when multiple criteria are considered.

Different studies focused on improving the performance of DRs by evolving DRs that are used simultaneously. One such approach is the GP-3 method proposed in (Jakobović and Budin, 2006), which evolves two DRs and an additional expression. The evolved expression determines, based on the current system characteristics, which out of the two evolved DR should be used. Although this approach obtained good results, it could not compete with approaches that were subsequently designed. In other studies different ensemble learning methods were used to evolve several DRs, which jointly performed the decisions on which job to schedule next (Park et al., 2015; Hart and Sim, 2016; Park et al., 2018; Durasević and Jakobović, 2017; Đurasević and Jakobović, 2019). Such ensemble learning methods have demonstrated the ability to significantly improve the performance of DRs in comparison to the case when only a single rule is applied. Since these methods can be used to create ensembles of arbitrary DRs, they represent one of the most flexible and efficient methods of improving the performance of DRs. Several studies have also researched how different solution representations affect the results. A multi-tree representation was used in (Zhang et al., 2018), where the rules to perform the sequencing and routing decisions were evolved simultaneously. This paper demonstrated that by splitting these decisions into two independent rules it was possible to improve the performance of DRs. In (Branke et al., 2015) three representations of DRs were tested, out of which GP and artificial neural networks evolved the best DRs. The main benefit of GP is that it evolves DRs that are interpretable, whereas the artificial neural networks are difficult to interpret. This further supports the choice of GP as the dominant method for creating new DRs. In (Nguyen et al., 2013) three solution representations for DRs were tested, one which is used to select existing DRs, one which evolved new DRs, and a combination of the two aforementioned methods. The solution which used only existing DRs achieved the poorest performance, which demonstrated that it is essential that GP can construct completely new DRs, or at least extend existing ones. In (Đurasević and Jakobović, 2020) it was demonstrated that DRs can be adapted for static scheduling and compete with other metaheuristic methods. This result is significant as it shows that, given the same conditions as metaheuristic methods, the evolved DRs obtain only slightly worse results, showing that these methods are equally expressive as search based metaheuristics.

The previous paragraphs outline that the research in automatically generating DRs covered a wide range of topics, both in extending the methods to different problems, but also in proposing methods for improving the performance of automatically generated DRs. Initial studies laid out the foundations in a sense that they outlined which representations should be used, which terminal or function nodes to apply and similar. Recent studies focused more on improving the performance of automatically generated DRs by proposing new methods, or applying the approach to new problem types. The advantage of most methods proposed in these studies is their flexibility, meaning that they can be applied for different problems variants. However, evolving DRs for scheduling problems with additional constrains has been researched sparsely. Most studies focused on the setup times constraint. In (Jakobović et al., 2007) setup times were considered for the parallel uniform machines environment. In this study the authors shortly considered the setup times, for which they propose two new terminal nodes. The results demonstrate that the obtained DRs performed better than existing manually designed DRs. In (Pitzer et al., 2011) the setup times were also considered, but only a single terminal that included information about setup times was used. As such, the constructed DRs had a poor overview of this constraint, but still managed to outperform existing DRs. In (Pickardt et al., 2013) the setup times were considered when creating DRs for work scheduling in semiconductor manufacturing. Again, only a single terminal that provided information about setup times was used, and no deeper analyses were performed on the influence of setup times on the performance of automatically designed rules. DRs were also evolved for the single machine environment with setup times in (Jakobović and Marasović, 2012), but unfortunately this study already used existing terminal nodes and did not bring new insights into this topic. Several papers dealing with the OAS problem considered setup times as well (Nguyen et al., 2013; Park et al., 2013), but here the main topic was the adaptation of GP for the OAS problem. In neither of the aforementioned papers a thorough analysis on the evolution of DRs for setup times was performed, since the focus was set on other topics. As a result it is not clear how different terminal nodes influence the results and if they are needed at all.

Several papers have also dealt with machine eligibility constraints (Beham et al., 2008; Tay and Ho, 2008; Nie et al., 2013) in the flexible job shop environment. However, neither of these studies did provide a detailed overview on that constraint and how it can affect the scheduling process. Additionally, no new terminal nodes were proposed that would include information about the eligibility of jobs. Thus, no new insights about the influence of this constraint can be gained from the previous studies. Regarding machine unavailability, it was considered in (Wen-Jun Yin et al., 2003), where the authors designed DRs with GP for the single machine environment. The authors introduced new terminal nodes and demonstrated that automatically generated DRs outperformed manually designed DRs since they could better adapt to the appearance of breakdowns in the system. However, no research was performed in more complex environments with several available machines. This study nevertheless represents a good initial study for that constraint. Finally, the precedence constraints were considered only in (Jakobović and Marasović, 2012) for the single machine environment. For that constraint two new terminal nodes were proposed. The obtained results were encouraging as they demonstrated that the generated DRs could handle these constraints. However, no deeper analysis or application to more complex environments was done after this initial study. The previous paper is also the only one which considered solving a problem consisting of a combination of two constraints, namely setup times and precedence constraints. Although good results were obtained even in this case, there was no follow up on this topic. Other constraint combinations were not even considered in the literature. Although some studies did take into account additional constraints, no study put its focus solely on them to determine how to best solve such problems. Additionally, most research was performed on the single machine environment, which is relatively simple. A small number of terminal nodes were proposed and their influence on the performance of automatically generated DRs was not analysed. Therefore, the main goal of this study is to fill this gap by considering the constraints in a more complex scheduling environment, testing different constraint combinations, proposing new terminal nodes and evaluating their effectiveness.

3. Unrelated machines scheduling environment

Scheduling problems are usually defined by a number of jobs which need to be scheduled on a given set of machines. The number of jobs and machines is usually limited and is denoted with n and m, respectively. Furthermore, a certain job in the system is usually denoted as j, whereas a machine is denoted as i. The unrelated machines environment represents the class of scheduling problems in which each job needs to be executed on only one machine, however, the time needed to execute the job depends both on the job and machine. Therefore, it is impossible to specify that some machines are faster than others, since one job can execute faster on one machine than the other, whereas for another job the opposite can be true.

The job properties that are defined for a scheduling problem vary depending on the problem under consideration. However, in all variants it is required to define the processing time of each job for each of the machines. This property is denoted as p_{ij}. When dynamic scheduling problems are under consideration, each job also has a release time r_j that denotes the moment in time when job j is released and can be scheduled. Each job can also have a weight w_j that specifies the importance of the job in a way that scheduling a more important job later on will induce a larger penalty value. Finally, in some problems it is important to complete the jobs until a certain time moment, which is called the due date d_j. If a job is not finished until its due date it will continue executing, but the longer it executes the greater the penalty which it incurs will be.

In addition to the aforementioned properties, depending on the additional constraints that are considered in the scheduling problem, further job and machine properties can be defined. In this paper the following constraints are considered:

- Sequence dependent setup times - define a certain amount of time that needs to be invested to prepare a machine for executing the next job. This constraint can be, for example, found in a printing press where it is required to setup the machine before something else can be printed (Pinedo, 2012). In this constraint a setup time s_{jk} is defined for all job pairs, which denotes the time that has to be invested to prepare the machine for executing job k, if prior to it job j was executing on the considered machine. If the job under consideration is the first which has to be scheduled on the machine, then the setup time is considered to be 0.

- Machine eligibility restrictions - specify that each job can be executed on a subset of available machines. Therefore, for each job j a set of eligible machines M_j, which can execute that job, is defined. This constraint can appear when scheduling programs on a cluster of computers, where it is possible to schedule the program only on those computers that have the required prerequisites, be it hardware or software.

- machine unavailability periods - specify that machines can be unavailable at certain periods of time. For example, a maintenance can be scheduled for each machine after it executed for a certain amount of time, or some machines can be unavailable in certain periods of time due to various reasons like maintenances. Therefore, for each machine a set of time intervals, which determine time periods when the machine is unavailable, is specified.

- Precedence constraints - define that one or more jobs have to be executed before another job can start its execution. For example, if several processes are executed on computers, one process might require the output of several others and cannot start executing until all processes upon which it depends have been completed. In this constraint, each job has an additional set which contains all jobs that have to be completed before the considered job can start executing.

The criterion which will be optimised is the total weighted tardiness (TWT), which is defined as $TWT = \sum_{j=0}^{n} w_j \max(C_j - d_j, 0)$, where C_j defines the time when job j finishes with its execution. The goal of this criterion is to minimise the time that jobs spent executing after their due date, with the while giving more importance to jobs with a higher weight.

Aside from the previously outlined properties of the scheduling problem, it is required to specify the conditions under which scheduling is performed. For most part in this paper the dynamic variant will be considered. In dynamic scheduling problems the information about jobs (like their processing or arrival times) are not known in advance, and becomes available only when the job is released. As a result, the schedule cannot be constructed before the start of the system, but rather has to be constructed in parallel with the system execution. On the other hand, in static scheduling all the information about the problem is known beforehand, and thus it is possible to construct the entire schedule even before the system starts with its execution. Additionally, the problems which will be considered in this paper are deterministic, meaning that the values of all properties are precisely known when they become available.

4. Solving scheduling problems with constraints

Although this paper deals with the automatic generation of DRs for the unrelated machines environment with additional constraints, two other ways of solving these problems will be applied. This is done to validate the results obtained by GP, since there are no relevant methods that could be used for solving problems with all the aforementioned constraints. Therefore, in addition to GP a GA will also be used to solve the static scheduling problem variant, which can be considered as a certain lower bound for the problems. This result is used to get a notion on where the results obtained by automatically designed DRs stand in comparison to improvement heuristics that search the entire solution space. In addition to that, five existing manually designed DRs were selected and adapted to support all the considered constraints. The reason why these rules were selected is due to the fact that they achieve the best results when considering the TWT criterion for the unrelated machines environment without constraints (Đurasević and Jakobović, 2018).

4.1. Genetic algorithm

The applied GA uses the machine list encoding (MLE), which which is one of the most popular encoding schemes for the unrelated machines scheduling problem. This encoding contains m lists, each of which represents one of the machines. Each list contains those jobs which should be executed on the corresponding machine in the order in which they appear in the list. As for the genetic operators, the point crossover and insertion mutation were used. The point crossover selects a random crossover point for each list and creates the child by combining the jobs of one parent which appear before the crossover point and the jobs which come after the crossover point from the other parent. The insertion mutation selects a job randomly and inserts it into a random list on a random position. More details about the encoding and the applied operators can be found in (Vlašić et al., 2019).

When additional constraints are used, different parts of the algorithm need to be adapted to ensure that feasible schedules are produced. Most notably, the generation of the initial population, the crossover, and mutation operator need to be adapted. For the setup times no modifications have to be made, since all schedules that can be constructed will be feasible. The situation is similar for the machine unavailability constraint. This constraint is satisfied during the evaluation of the solution. Namely, if a job execution would overlap with an period of machine unavailability, the execution of the job is postponed to a later time when the job can execute completely without any interruption. Machine eligibility constraints need to be incorporated in the initialisation process and genetic operators in a way that a job can be allocated only to an eligible machine. On the other hand, for precedence constraints it is more difficult to ensure the feasibility of solutions. In case of this constraint, each time a job is scheduled on a machine it needs to be ensured that all predecessors of this job are scheduled prior to it. Furthermore, it is required that no deadlock appears in the schedule, i.e. that a situation in which some scheduled jobs cannot start executing because their predecessors were not previously executed. It is quite complicated to ensure this, and it also causes the operators to be less time efficient since they have to determine which changes can be made to keep the schedule feasible.

4.2. Adaptation of manual DRs

To deal with the considered constraints, the original manually designed DRs need to be adapted. The simplest way to adapt existing rules for various constraints is by modifying their schedule generation scheme (SGS) to ensure that the constructed schedules satisfy all given constraints. Since the considered DRs use the same SGS, with the only difference being the priority function $I_{ij}(t)$ that is used to calculate the priorities for jobs, only a single adapted SGS has to be defined. Algorithm 1 denotes the extended SGS for manual DRs. The only difference over the original SGS that these rules use is in the addition of the instruction in line 10, which checks whether all the conditions for a feasible schedule are met. Only then the priority value is calculated, otherwise the machine and job under consideration are skipped since they would lead to the construction of an unfeasible schedule. The conditions that have to be checked are that the machine is eligible for job j, that no unavailability will occur during the execution of job j, and that all predecessors of job j have finished with execution. As can be seen from the description, the only constraint that is not considered are setup times, which is because they have no influence on schedule feasibility and as such it is not required to consider them explicitly in the SGS.

Algorithm 1 SGS for manually designed DRs adapted for additional constraints

```
 1: $n \leftarrow$ number of jobs
 2: $m \leftarrow$ number of machines
 3: $schedule[m][n] = []$
 4: while all jobs are not scheduled do
 5:     $t \leftarrow$ current system time
 6:     $unscheduledJobs \leftarrow$ available unscheduled jobs in the system
 7:     while $unscheduledJobs$ not empty do
 8:         // find the job with the best priority
 9:         for all $j \in uscheduledJobs, i \in m$ do
10:             if job $j$ is eligible for machine $i$ and machine $i$ will not become
11:             unavailable during its execution and all predecessors of job $j$ have
12:             finished executing then
13:                 $priority \leftarrow I_{ij}(t)$
14:             end if
15:         end for
16:         $job \leftarrow$ job with the highest priority
17:         $machine \leftarrow$ machine on which $job$ will execute the soonest
18:         $schedule[machine] \xleftarrow{\text{if machine is available}} job$
19:     end while
20: end while
21: $fitness \leftarrow$ evaluate $schedule$
```

The priority function that is used depends then on the concrete DR. For the five selected DRs they are equal to:

- EDD - $I_{ij}(t) = \frac{1}{d_j}$
- MS - $I_{ij}(t) = -\max\left(d_j - p_{ij} - time, 0\right)$
- MON - $I_{ij}(t) = \frac{w_j}{p_{ij}}\left(1 - \frac{d_j}{p_s}\right)$
- COV - $I_{ij}(t) = \frac{w_j}{p_{ij}} \max\left[\left(1 - \frac{\max\left(d_j - p_{ij} - time, 0\right)}{k\bar{p}}\right), 0\right]$
- ATC - $I_{ij}(t) = \frac{w_j}{p_{ij}} \exp\left[-\frac{max(d_j - p_{ij} - t, 0)}{k_1\bar{p}}\right] \exp\left[\frac{s_{lj}}{k_2\bar{s}}\right]$,

where t represents the current time of the system, $\hat{p}$ represents the average processing time of all unscheduled released jobs, p_s represents the sum of processing times of all available jobs for machine i, $\hat{s}$ represents the average setup times of all released and unscheduled jobs, while k, k_1, and k_2 represent control parameters that are defined by the user. The priority function used for ATC is the adapted one which also considers setup times if they are present (Lee et al., 1997). For the other three criteria it would also be possible to extend the priority functions with additional terms. However, extending the priority function in a meaningful way is not easy. Some initial attempts were made in this direction as well, but the additions did not result in any improved results. Therefore, it was decided to keep the changes only in the SGS.

4.3. Designing DRs with GP

Aside from manually designing new DRs, or extending existing ones, it is also possible to use GP to evolve new DRs automatically. More specifically, GP is used to generate the priority function (PF) that determines the priorities of jobs and machines, whereas the SGS that creates the complete schedule is defined manually. The reason for this is that a meaningful SGS can be more easily defined manually than it can be evolved automatically. On the other hand, it is much harder to design the PF manually, whereas GP and similar methods can easily obtain PFs which perform well.

Algorithm 2 represents the SGS which is used with automatically designed PFs. This SGS works in a way that each time a scheduling decision has to be performed (scheduling a job on a machine), the SGS calculates the priority of each available job on all machines. For each of the available jobs the SGS determines the machine for which the job obtained the best priority value. Then, the jobs are scheduled on the selected machines in the order of their priorities, but only if the machine is available. Otherwise, the job is skipped and will be scheduled later. Naturally, the SGS is also enhanced to take into consideration all the constraints. First, the precedence constraints are taken into account in line 4, where the priorities only for those jobs with all finished predecessors are calculated. If all predecessors of a job are not executed, then the job will have to wait until all predecessors have been executed. The machine eligibility constraint is enforced in line 6, by calculating the priorities of jobs only for those machines which are eligible for the considered job. In that way, a job can be scheduled only on eligible machines. Finally, the machine unavailability constraint is considered in line 17, where it is checked whether the selected machine would be unavailable during the execution of the selected job. If it would, the job will not be scheduled on this machine at the current time. One might wonder why this constraint is not checked at the beginning of the SGS, together with the other two constraints, since in the way the SGS is defined it is possible that a job will have the highest priority, but will not be scheduled due to machine unavailability. However, in this way the SGS and GP have more flexibility, since GP can design a PF which takes into account the availabilities and can decide whether it makes more sense to schedule a job on a machine which will be available, or to wait until the unavailability of a certain machine ends and then schedule the job on that machine.

Algorithm 2 SGS for automatically designed DRs adapted for additional constraints

```
1:  while unscheduled jobs exist do
2:      Wait until a job or machine becomes available
3:      for all available jobs $j$ do
4:          if all predecessors of job $j$ have finished executing then
5:              for all each machine $i$ do
6:                  if machine $i$ is eligible for job $j$ then
7:                      Calculate the priority $\pi_{ij}$ of scheduling job $j$ on machine $i$
8:                  end if
9:              end for
10:         end if
11:     end for
12:     for all available jobs $j$ do
13:         Determine the machine $m_j$ for which job $j$ obtained the best $\pi_{ij}$ value
14:     end for
15:     while there are jobs whose selected machine $i$ is available and do
16:         Determine the job $j$ with the best priority
17:         if machine $m_j$ will be available for the entire execution of job $j$ then
18:             Schedule job $j$ on the machine $m_j$
19:         end if
20:     end while
21: end while
```

GP is used to evolve a PF that is in turn used by the aforementioned SGS. In order to apply GP, a set of primitive nodes needs to be defined, which serve as building blocks for the creation of PFs. For function nodes the following five are used: addition, subtraction, multiplication, secure division (1 is returned in case of division with 0), and the positive operator defined as $pos(x) = max(x, 0)$. Although other function nodes can be used, previous research did not show that better results can be obtained by using additional function nodes (Đurasević et al., 2016).

Aside from the function nodes, a set of terminal nodes also has to be defined. These nodes represent certain job, machine, and system properties, which are combined into meaningful expressions in GP. Table 1 lists the basic set of terminal nodes, which represent only general system information (like job processing times, weights, and similar), but does not include any information about the additional constraints that are considered. This set represents the base terminals which are used in all experiments and extended with additional terminal nodes, depending on the constraint.

In addition to the previously outlined general terminal nodes, Table 2 additional nodes which take different constraints into account. They are divided into four groups depending on the constraint they were designed for. In case

Table 1: Terminal set

Terminal	Description
pt	processing time of job j on machine i (p_{ij})
$pmin$	minimal processing time of job j
$pavg$	average processing time of job j on all the machines
PAT	time until machine with the minimum processing time for job j becomes available
MR	time until machine i becomes available
age	time which job j spent in the system
dd	time until which job j has to finish with its execution (d_j)
w	weight of job j (w_j)
SL	slack of job j, $-max(d_j - p_{ij} - t, 0)$

of the setup times, the additional terminal nodes provide the basic information about the setup times of jobs, like the minimal setup time, average setup time, and the setup time for the chosen machine. By using these nodes the PF can determine on which machine the setup time would be the lowest and would be most suitable for scheduling the selected job. For the machine eligibility constraint the nodes provide the information about the number of eligible machines for each job, the number of available machines that can execute a job, and the number of released jobs that are eligible for a certain machine. In that way the PF can take into account how many machines are eligible for a job, e.g. to be able to schedule sooner those jobs for which only a few machines are eligible. For machine unavailability four terminal nodes are defined. These nodes provide information about the remaining unavailability periods, the time until the current unavailability ends, the time until a machine becomes unavailable again, and whether the machine will become unavailable during the execution of a job. With these terminals the PF can determine whether the machine will become unavailable during the execution of a job and give a smaller priority to such jobs. Additionally, the PF can also determine that, if an unavailability period will end soon, it could be more profitable to wait for it to end and schedule the considered job on that machine rather than to schedule it immediately on a different machine. Finally, since the precedence constraints are the most complicated, six terminal nodes were defined for them. These nodes provide the number of successors of a job, but also more complex information like the slack values of the released successors. The goal of these more complex terminal nodes is to give the PF a notion on how selecting one or the other job could possibly influence the criterion value by approximating the possible slack and tardiness of all the successors of a job.

Table 2: Additional terminals for the considered constraints

Terminal	Description
	Terminals for setup times
smin	minimal setup time of job j
sAvg	average setup time of job j
setMac	setup time of job j on job i
	Terminals for eligibility constraints
emfj	number of machines which are eligible for job j
rjfm	number of released jobs that can be executed on machine i
amfj	number of available machines that can execute job j
	Terminals for machine unavailability periods
bwhde	$\begin{cases} 1, & \text{if machine will be unavailable during the execution of job j} \\ 0, & \text{if machine will be available during the execution of job j} \end{cases}$
norb	number of remaining unavailability periods
tube	time until the unavailability ends
tunb	time until the next unavailability
	Terminals for precedence constraints
nous	number of released successors of job j
avss	average *slack* of all released successors of job j
mss	minimum *slack* of all released successors of job j
mxss	maximum *slack* of all released successors of job j
wsl	weighted tardiness of all released successors of job j
scl	number of all released successors (direct and indirect) of job j
mscl	the number of released successor jobs in the longest chain of job j

5. Benchmark setup

5.1. Problem instance design

To validate the proposed methods, a set of problem instances was designed in a similar way as in several other studies (Dimopoulos and Zalzala, 1999, 2001; Lee et al., 1997; Pfund et al., 2008). The instances were generated with a different number of jobs and machines in various combinations. The number of jobs was set to 12, 25, 50, or 100, whereas the number of machines was set to 3, 6, or 10.

The processing times of jobs are generated from the interval

$$p_{ij} \in [0, 100].$$

The distributions used to generate the processing time from this interval are the uniform, Gaussian, and quasi-bimodal. Each processing time is generated from the specified interval by randomly using one of the three aforementioned distributions. In that way it is simulated that jobs of different sources, which behave differently, are released into the system. Job weights are generated uniformly from the interval

$$w \in (0,1].$$

Job release times are generated by a uniform distribution from the interval

$$r_j \in \left[0, \frac{\hat{p}}{2}\right],$$

where $\hat{p}$ is defined as

$$\hat{p} = \frac{\sum_{j=1}^{n} \sum_{i=1}^{m} p_{ij}}{m^2},$$

and p_{ij} denotes the processing time of job j on machine i, while m denotes the total number of machines. The job due dates are generated using a uniform distribution from the interval

$$d_j \in \left[r_j + (\hat{p} - r_j) * \left(1 - T - \frac{R}{2}\right), r_j + (\hat{p} - r_j) * \left(1 - T + \frac{R}{2}\right)\right],$$

where T represents the due date tightness parameter, while R denotes the due date range parameter. The values of those parameters were set to 0.2, 0.4, 0.6, 0.8, and 1 in various combinations.

The setup times, s_{ij} were generated uniformly from the following interval:

$$s_{ij} \in [0, s_{max}]$$

, where s_{max} represents the maximum allowed setup time. For the experiments, the s_{max} parameter was set to 5.

For the generation of machine unavailability periods several parameters have to be specified. First, for each machine a number of unavailability periods has to be defined. The number of unavailability periods per machine ranged from 0 to 12, in a way that smaller problem instances had a smaller number of unavailability periods, whereas the larger instances had more unavailability periods. The start of each unavailability period is generated from the interval $[0, \hat{p}]$. The duration of the unavailability period is uniformly generated from the interval $[u_{min}, u_{max}]$, where u_{min} represents the minimum unavailability duration, whereas u_{max} represents the maximum unavailability duration. The values for u_{min} and u_{max} were set to 1 and 10, respectively.

The machine eligibility constraints are generated in a way that for each machine a percentage of jobs that are eligible for it can be specified. This percentage ranged from 0.5 to 1 depending on the number of machines. The jobs that are eligible for each machine are randomly selected until the specified percentage of jobs is reached for each machine. In addition, it is also ensured that each job has at least one machine which is eligible for it, so that it is possible to create a feasible schedule.

Finally, for the precedence constraints three parameters were defined. First, the parameter which defines the percentage of jobs that will have a predecessor. This parameter was set to 0.2 and 0.3 depending on the problem instances. In addition to those two parameters, the maximum number of predecessors and successors a job can have is also defined. These were set between 2 and 10, depending on the number of jobs in the problem instance.

Because some problem instances have significantly different characteristics, they also have objective values of different scales. This leads to a problem in which smaller instances have little or no influence in the total fitness value and thus the GP procedure would focus less on optimising these instances. To avoid this problem all the objective values were normalised in order for the problem instances with different characteristics to have similar objective values. Therefore, the normalised objective functions for the problem instance with the index i is defined as

$$f_i = \frac{\sum_{j=1}^{n} w_j T_j}{n \bar{w} \bar{p}}$$

where n denotes the number of jobs in the problem instance, $\bar{w}$ the average weight of the jobs and $\bar{p}$ the average job processing duration. The total objective function is then calculated as the sum of the objective functions of the individual problem instances.

5.2. Experimental setup

To test the proposed methods for optimising the unrelated machines scheduling problem with constraints, two problem instance sets were generated as described in the previous subsection. The first problem instance set, called the training set, is used by GP to evolve new PFs. The second problem instance set, denoted as the test set, is used after the training period to evaluate the evolved PFs on unseen problem instances and to determine how well they perform. Both sets contain 60 individual problem instances.

To obtain statistically significant results, each experiment was executed 30 times, and the best individual in each execution was saved. For these 30 executions the minimum, median, and maximum values are calculated. Furthermore,

Table 3: ATC parameter values

setup	●				●	●	●				●	●	●		●
eligibility		●			●			●	●		●	●		●	●
unavailability			●			●		●		●	●		●	●	●
precedence				●			●		●	●		●	●	●	●
k_1	0.95	0.45	1.5	1.3	0.6	1.9	0.8	0.2	0.4	0.6	0.6	0.4	1.9	0.7	0.5
k_2	0.1	-	-	-	1.9	1.9	1.2	-	-	-	1.9	0.5	1.4	-	1.1

the Kruskal-Wallis statistical test is used to determine whether a statistically significant difference exists between a group of experiments. If a significant difference was obtained, the Canover p-values adjusted with the Benjamini-Hochberg FDR method are further calculated to determine between which pairs of experiments a significant difference exists. Additionally, the Mann-Whitney statistical test was also used to perform pairwise comparisons between the algorithms in several situations of interest. For both tests the difference between results is denoted statistically significant if the test obtained a p-value lower than 0.05.

Finally, the parameters of all the methods were optimised in preliminary experiments and the best obtained parameters were used. For the GP and GA the parameters were optimised only once since the parameter optimisation is quite expensive. Both GA and GP use a steady state tournament selection. The GA uses a population size of 30 individuals, a mutation probability of 0.9, the tournament size of 3 individuals, and a termination criterion of 1000000 function evaluations. The point crossover and insert mutation were used by the GA. On the other hand, GP uses a population size of 1000 individuals, a mutation probability of 0.3, the tournament size of 3 individuals, the ramped half-and-half initialisation method, a maximal tree depth of 5, and a termination criterion of 80000 function evaluations. For crossover the subtree, uniform, context-preserving, and size fair operators were used (Poli et al., 2008). For mutation the subtree, Gauss, hoist, node complement, node replacement, permutation and shrink operators were used (Poli et al., 2008). Out of the selected manually designed DRs only the COV and ATC rules have adjustable parameters. For the ATC rule, the two control parameters, k_1 and k_2, were optimised. Since the performance of the ATC rule depends heavily on these two parameters, they were optimised for each combination of constraints that was tested. The values of these two parameters for each of the experiments are denoted in Table 3. The COV rule uses only a single parameter and preliminary experiments demonstrated that the rule performed best when this parameter is set to 0.05.

6. Results

In this section the results of all the methods for the various combinations of constraints will be outlined. First, each constraint will be optimised individually to determine the effect of the terminal nodes on the performance of automatically designed DRs. After that, the best combinations of terminal nodes will be used when several constraints are optimised simultaneously.

6.1. Optimisation of individual constraints

6.1.1. Setup times

The results obtained for the scheduling problem with setup times are denoted in Table 4. The results demonstrate that the GA performs better than any of the tested DRs. Such a behaviour is expected in static scheduling since the GA has access to all the information of the problem while the other methods operate under dynamic conditions. Out of the tested manually designed DRs, the best results were obtained by the EDD and COV rules. It is surprising that the ATC rule which was adapted for problems with setup times obtained the worst results. The main reason for this is that poor values for parameters k_1 and k_2 were selected. This shows that the ATC rule is sensitive to the choice of parameter values, even though the training set on which they were optimised is similar to the test set. The DRs evolved by GP easily outperform the selected manually designed DRs. Even in the case when no additional terminal nodes are used, the GP evolved DRs perform significantly better than any of the tested rules. This can best be seen from Figure 1, which shows that most of the automatically designed DRs performed better than manually designed DRs. It is interesting that even without any additional nodes the evolved DRs achieve a better performance. The reason for this is that GP can create PFs that are better adapted for the concrete set of problems. Based on the previous observations, it is possible to conclude that for this constraint the evolved DRs are less sensitive to the problems that are solved, unlike the ATC rule whose performance largely depends on the selected parameter values.

When the GP variants with different terminal nodes are compared with each other, it is evident that all the variants achieve quite similar results. All the experiments in which GP used additional terminal nodes achieve a better median value than the experiment in which no additional nodes were used. This shows that the proposed nodes do provide useful information which the automatically generated DRs can use. However, the statistical tests show that the difference is not significant in all cases. The Kruskal-Wallis test obtained a p-value of 0.046, which means that a statistically significant difference between the experiments exists. Further tests have a shown that a significant difference exists only between the results obtained when no additional terminals are used and when the combination of the *savg* and *setmac* terminals is used. In this case, the experiments obtained when using no additional terminals were significantly worse than those obtained when using the two additional terminal nodes, with a p-value of 0.034. Therefore, only one combination of nodes really lead to the improvement of the results. One reason why the new terminals did lead to improvements, but which were not significant, could be due to the reason that setup times did not have a large enough influence in the problem instances. This combination leads to the best results probably because of the good synergy these

Table 4: Results obtained when considering the setup times

	min	med	max
GA	10.73	11.04	11.36
EDD		19.92	
MS		21.05	
MON		21.42	
COV		19.89	
ATC		21.83	
GP			
no additional terminals	17.52	18.84	20.32
smin	17.06	18.47	23.37
sAvg	17.10	18.42	20.01
setMac	17.08	18.40	21.66
smin, *sAvg*	16.83	18.36	20.21
smin, *setMac*	16.66	18.36	20.16
sAvg, *setMac*	16.97	18.40	19.45
smin, *sAvg*, *setMac*	16.88	18.32	20.25

two nodes provide. The *setmac* node gives the information about the length of the current setup time, and the *savg* allows the rule to compare how the current setup time compares to the average of all setup times for the job.

The results demonstrate that for this constraint it makes sense to use additional terminal nodes. Although the results might not be significantly better for all the cases when they are used, the evolved DRs perform better. The reason why for this constraint additional terminal nodes are useful is because the setup times affect the results obtained for the optimised criterion, and therefore if the processing times of all jobs are similar it becomes important to take the setup times into account when determining which job to select and on which machine to schedule it. Without these nodes the PF does not have access to this information and cannot take setup times into consideration when scheduling jobs. Since all the combinations obtained quite similar results, it is difficult to outline which terminal nodes are the most informative to the PFs. Based on the statistical tests it seems that those would be the *sAvg* and *setMac* nodes. The reason why the *sMin* node does not seem to be informative is because by itself it gives little information about the setup times. It is also interesting to observe that better results were always obtained when more nodes were used simultaneously. This is expected as in this cases the DRs have access to more information and can perform decisions based on a combination of these individual terminal nodes.

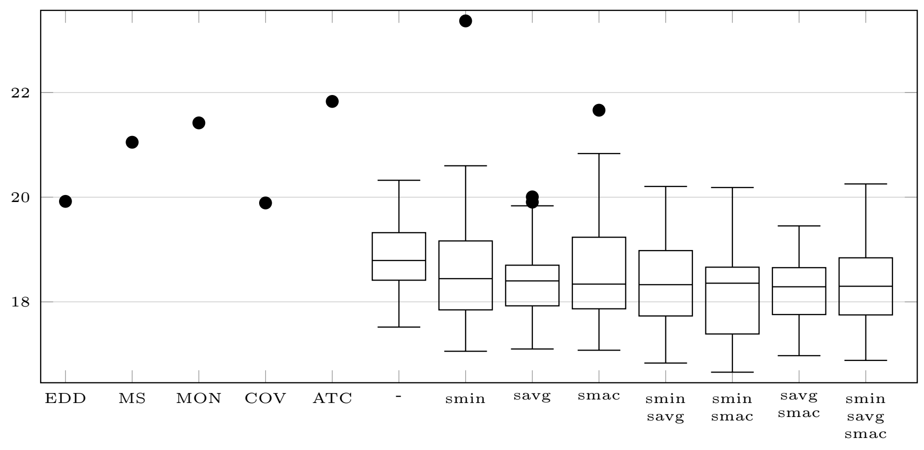

Figure 1: Box plot representation of results obtained when considering setup times

6.1.2. Machine eligibility

The results obtained when considering the machine eligibility constraint are denoted in Table 5. The GA again achieves the best results. However, the difference between the results obtained by the GA and automatically designed DRs is smaller in comparison with experiments where setup times were considered. The reason behind such a behaviour is because for setup times the solution space does not decrease. Thus, the problem becomes more complicated since in addition to processing times the setup times are also considered. For eligibility constraints the situation is different, since they reduce the number of machines on which a job can be executed. This means that the evolved DRs will have fewer choices to choose from when scheduling a job. Therefore, there is a higher chance that DRs will perform better. The table also shows that the manually designed DRs obtain inferior results compared to the automatically designed rules. The best result from those rules is obtained by ATC, whereas the others achieve inferior results. Even when no additional terminal nodes are used the evolved DRs obtain better results than the manually designed rules. Figure 2 shows that the ATC rule performs better than a few manually evolved DRs (which usually represent outliers), whereas all other manually designed DRs do not outperform even one automatically designed rule. This demonstrates that GP can easily evolve DRs that perform better than several manually designed DRs for this constraint.

When comparing the results obtained by the automatically designed DRs, one can observe that the median performance is similar for almost all combinations of additional terminal nodes and the case when no additional nodes were used. This is backed up by the Kruskal-Wallis test which obtained a p-value of 0.066, which means that the results are not significantly different. However, figure 2 shows that some node combinations influence the performance. For example, using either the node *amfj* or *emfj* leads to less distributed results.

Table 5: Results obtained when considering the machine eligibility

	min	med	max
GA	14.58	15.15	15.94
EDD		27.07	
MS		30.56	
MON		22.87	
COV		22.63	
ATC		20.30	
GP			
no additional terminals	17.49	18.33	20.93
amfj	17.61	18.27	21.89
emfj	17.43	18.26	20.20
rjfm	17.63	18.47	21.03
amfj, *emfj*	17.44	18.54	21.27
amfj, *rjfm*	17.54	19.21	20.86
emfj, *rjfm*	17.37	18.38	20.86
amfj, *emfj*, *rjfm*	17.64	18.77	21.41

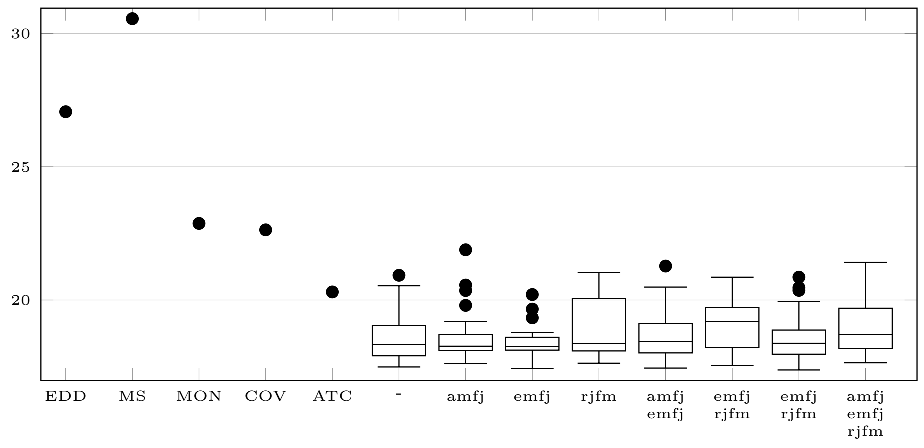

Figure 2: Box plot representation of results obtained when considering machine eligibility

Therefore, these nodes seem to contain useful information that GP evolved rules can exploit. However, this information was still not enough to obtain significantly better results. On the other hand, some other combinations lead to heavily dispersed results. Interestingly, using the *amfj* and *emfj* nodes at the same time leads to worse results then when using them individually. This shows that combining terminals which perform well by themselves does not necessarily lead to improved results. Even though the proposed terminal nodes did not result in significantly better results, their use can still prove to be viable since they reduce the dispersion of results and improve the chances of obtaining a better DR.

As already outlined, the results show that no significant improvement was achieved by using additional terminal nodes, which for this constraint makes sense. This is due to the fact that this constraint is mostly handled by the SGS. Namely, the SGS ensures that the PF is evaluated only for eligible machines. Thus, the SGS actively restricts the number of possible decisions and prevents the construction of unfeasible schedules. In this situation, the DRs do not even need any additional terminal nodes, because this constraint does not need to be considered in the PF. The only situations in which additional terminals could prove useful would be when a job can be scheduled only on a single machine. In that case those nodes could help the PF to detect such jobs to schedule them immediately when their eligible machine becomes free. This is backed up by the results which show that out of the terminal nodes that were defined for this constraint, the *emfj* and *amfj* were the ones which slightly improved the results. This is because they allow the DRs to determine jobs which can be scheduled only on few machines to prioritise them. However, such situations are rare, and as such these nodes will only slightly improve the performance of DRs.

6.1.3. Machine availability

The results obtained when considering the machine availability constraint are denoted in Table 6. The GA again obtained much better results than it was possible to obtain with any of the tested DRs. The adapted manually designed rules obtain the worst results amongst all the tested methods. The ATC rule again obtained the best results among the manually designed DRs. Figure 3 shows that most automatically designed rules outperform the manually designed rules. DRs which were generated without using additional terminals largely outperform the manually designed DRs. The margin between their performance becomes larger when additional terminal nodes are used. It should be noted that for the combination of the *bwhde* and *tunb* nodes, every DRs that was evolved performs better than any manually designed rule. This demonstrates that by manually adapting existing rules it is difficult to match the performance that can be provided by the DRs that are automatically evolved by GP, even when not using any additional information about the constraint.

The results show that most node combinations obtain quite similar results. All the experiments in which additional terminal nodes were used achieved a smaller median value than the experiment in which no additional terminal nodes were used. Therefore, for this constraint it is important to include additional information about it in the form of terminal nodes. The Kruskal-Wallis test reported a p-value of 0.059, which would mean that there is no statistically significant difference between the various combinations. However, the pairwise Mann-Whitney test was also used, and showed that there exists a significant difference between the results obtained for the experiment which does not use additional terminal nodes, and the experiments which use the *bwhde* and *tube*, as well as the *hwhde* and *tunb* nodes. Therefore, it seems that some node combinations still lead to significantly better results when comparing them directly to the results with no additional terminal nodes.

For this constraint it was demonstrated that including additional information about the constraint into the PF leads to better results. This happens because the constraint does not introduce any additional restriction on which jobs can be scheduled or which machines can be selected. The reason for this is that if the selected job would execute during a period when the machine is unavailable, then the SGS would simply postpone that job to a latter moment in time, similarly as if the machine was busy. Therefore, the number of decisions that the DR can make are not reduced. Rather, more emphasis is set on the selection of the appropriate job or machine. One such choice would be if a job should wait for the unavailability to finish, or if it would be better to schedule it immediately on another machine if it is free. Without additional terminal nodes such a decision cannot be performed. Therefore, including these nodes into the PF gives it a much better overview of the problem. The most informative node seems to be the *bwhde* node, since in a combination with any other node it resulted in GP obtaining the best results. This node provides the information if an unavailability period appears during the execution of a job, which makes it pivotal in deciding whether to select a job or not. The results demonstrate

Table 6: Results obtained when considering the machine availability

	min	med	max
GA	11.35	11.69	11.89
EDD		26.70	
MS		25.75	
MON		23.11	
COV		22.91	
ATC		22.53	
GP			
no additional terminals	18.54	20.95	25.50
bwhde	18.27	20.53	22.87
norb	18.56	20.37	23.80
tube	17.99	20.19	25.62
tunb	18.55	21.34	25.90
bwhde, *norb*	17.71	19.97	25.11
bwhde, *tube*	18.02	19.78	22.31
bwhde, *tunb*	18.40	19.78	24.31
norb, *tube*	18.89	20.06	23.15
norb, *tunb*	18.38	20.10	23.42
tube, *tunb*	18.13	20.28	24.06
norb, *tube*, *tunb*	18.02	20.09	23.50
bwhde, *tube*, *tunb*	18.00	20.04	24.01
bwhde, *norb*, *tunb*	18.47	20.24	26.93
bwhde, *norb*, *tube*	18.40	19.96	24.82
bwhde, *norb*, *tube*, *tunb*	18.26	20.06	25.17

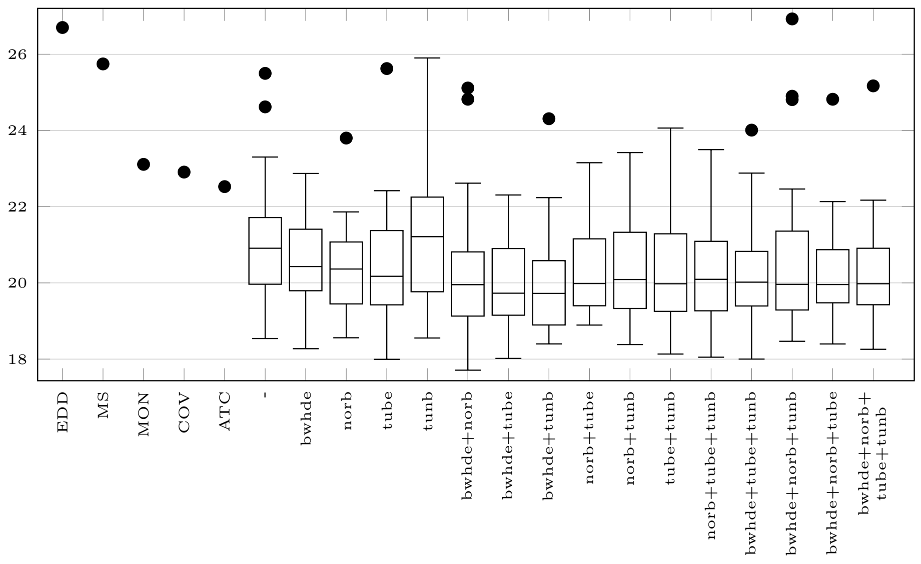

Figure 3: Box plot representation of results obtained when considering machine availability

that the performance of the DRs increases when at least two nodes are used simultaneously. This means that any single node by itself cannot provide enough information about the constraint. In combination with the *bwhde* node the best results were obtained when either the *tube* or *tunb* nodes were used. This is due to the reason that these two provide information about the time intervals of the current unavailability, which can have a more direct effect on the decision in comparison to the *norb* node, which just provides the information about the number of remaining unavailability periods.

6.1.4. Precedence constraints

Table 7 denotes the results obtained for problems with precedence constraints. Out of the manually designed DRs the ATC rule obtained the best results. When compared to the results obtained by the GP evolved DRs, all manually designed rules except ATC performed worse than most automatically designed ones. The result obtained by the ATC rule were better than the median value of all the experiments performed for designing new DRs, with or without using additional terminal nodes. This shows that most of the automatically designed DRs were unable to outperform the ATC rule. However, for all the tested node combinations it was still possible to obtain rules that outperform the ATC rule, but less frequently then for any of the previous three constraints. It is interesting that even when no additional nodes were used the automatically generated DRs usually performed worse than the ATC rule. As such it seems that even without additional terminal nodes GP struggles to obtain a good DR for this constraint. Based on these it is safe to conclude that this constraint represents a challenge when automatically designing DRs.

Table 7: Results obtained when considering job precedences

	min	med	max
GA	14.29	15.74	16.81
EDD		22.14	
MS		24.57	
MON		22.92	
COV		21.57	
ATC		20.20	
GP			
no additional terminals	19.30	20.69	26.71
nous	18.78	20.74	22.91
ass	19.37	20.71	27.61
mss	18.76	20.64	24.92
mxss	18.74	21.03	23.25
wsl	18.99	20.78	23.12
scl	19.28	20.74	25.42
mscl	18.74	20.96	23.17
ass, mss	18.83	20.50	24.37
$ass, nous$	19.03	20.64	24.86
ass, wsl	18.53	20.70	24.60
$mss, nous$	18.81	20.33	22.45
mss, wsl	19.30	21.09	28.27
$nous, wsl$	19.08	21.25	23.75
$ass, mss, nous$	19.54	20.73	24.78
ass, mss, wsl	18.64	20.64	22.76
$ass, nous, wsl$	19.00	20.45	23.78
$mss, nous, wsl$	19.66	20.98	23.62
$ass, mss, nous, wsl$	19.69	20.82	22.68

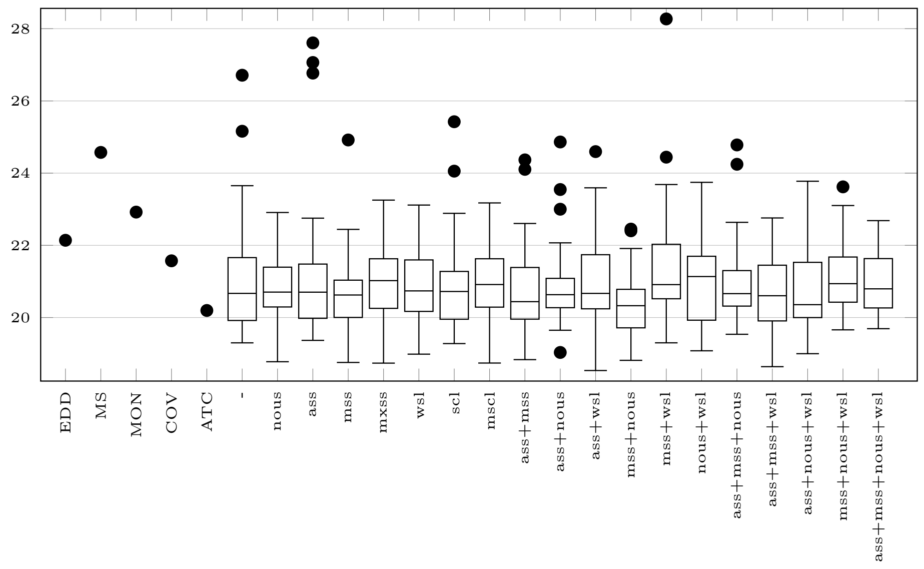

Figure 4: Box plot representation of results obtained when considering precedences

Figure 4 shows that there is little difference between the different experiments. For this constraint, the experiments that were performed by using additional terminal nodes quite often obtained a worse median value from the experiment in which no additional nodes were used. The p-value that was obtained by the Kruskal-Wallis test was equal to 0.26, which denotes that there is no significant difference between the obtained results. Although the median values obtained by the experiments with additional terminal nodes are usually worse than that of the experiment with no additional nodes, the same does not hold for the obtained minimum values. The minimum values show that for many combinations it was possible to obtain results that are better than the minimum value obtained when not using any additional terminal nodes. Thus, it seems that the additional terminal nodes do provide information that can be useful. However, for GP it seems to be quite difficult to find such rules. This could quite possibly be due to the fact that this constraint is more complicated than the previous one, and thus it is harder to incorporate meaningful knowledge about it into the designed DRs.

Regarding the terminal nodes that were used, it is difficult to outline which node seems to provide the most information to the DRs. Since the combination of the *mss* and the *nous* nodes obtained the best results, it is safe to assume that the information that these nodes provide is useful to the DRs. The *nous* node provides the information about the released successors, which allows the DR to prioritise those jobs that have many successors, since they would be blocking them from executing. On the other hand the *mss* provides the information about the slack of the successors of a job, and can also be used to determine which job should be prioritised. However, since the results are quite similar, it is not possible to make conclusive deduction.

6.2. Optimisation of combinations of constraints

Aside from considering each scheduling constraint individually, all possible constraint combinations are also examined to obtain a more thorough notion on how the automatically designed DRs perform in comparison with the manually designed rule. The results obtained for these combinations are denoted in Tables 8 and 9.

First of all, for all combinations the GA easily outperforms both the manually and automatically designed DRs. The differences are usually smaller when the constraints which reduce the number of decision in DRs are considered (for example, when the precedence times and machine eligibility constraints are considered together). On the other hand, the difference becomes larger for the constraints which introduce no restrictions in the decisions that cab be performed by DRs (like the combination of the setup times and machine unavailability periods). This means that it is more difficult to design DRs for constraints which do not limit the number of decisions that can be performed at each decision point.

When comparing the manually designed rules with the automatically designed DRs, in almost all the situations manually designed rules cannot achieve the same performance as the automatically designed DRs. The only rule which performed better than automatically designed DRs is the ATC rule, but only in two situations. The first was for the combination of setup times and precedence constraints. In this case the ATC rule achieved a slightly better result (by around 2.5%) than the median values that were obtained by automatically designed DRs. However, GP still found rules that were better than ATC (which can be seen by the minimum values). This is probably due to the precedence constraints for which it was already demonstrated that automatically designed rules had difficulties to obtain good results. The second situation in which ATC performed better was the combination of setup times, machine unavailability and eligibility constraints. In this case the difference is larger and more significant, around 11%. But even in this case GP obtained some rules which outperform ATC. This means that GP can deal with this constraint as well, but the problem might be in the selected terminal nodes and the algorithm struggled to obtain good DRs. However, one still has to take into consideration that the parameters of the ATC rule were optimised for each constraint combination, whereas the nodes in the GP were not. Therefore, by a more thorough selection of the nodes it could be possible to obtain even better results.

In all other cases automatically designed DRs outperformed the ATC and all the other manually designed DRs. In one occasion (when setup times and machine eligibility were considered together) the improvement was quite small, around 3%. However, in all other cases the improvements were more significant, ranging from 5% to 34%. The difference between the results obtained by manually and automatically designed DRs becomes larger as the number of constraints increases. Therefore, it is safe to conclude that as the number of considered constraints increases, it becomes increasingly difficult for manually

Table 8: Results obtained for combinations of two scheduling constraints

		min	med	max
machine unavailiability machine eligibility	GA	19.71	20.03	20.55
	EDD		49.29	
	MS		50.22	
	MON		43.17	
	COVERT		42.76	
	ATC		40.90	
	GP - standard terminals	30.50	34.09	85.97
	GP - additional terminals	28.80	33.05	39.04
setup times machine unavailability	GA	12.83	13.11	13.58
	EDD		34.98	
	MS		36.60	
	MON		31.12	
	COVERT		32.13	
	ATC		30.05	
	GP - standard terminals	23.76	26.07	30.62
	GP - additional terminals	23.26	25.27	29.59
setup times machine eligibility	GA	16.77	17.45	18.34
	EDD		38.23	
	MS		41.56	
	MON		31.51	
	COVERT		34.15	
	ATC		27.61	
	GP - standard terminals	23.82	26.75	30.97
	GP - additional terminals	24.26	27.58	46.53
machine unavailability precedence constraints	GA	15.84	19.10	20.50
	EDD		45.30	
	MS		49.65	
	MON		40.02	
	COVERT		63.90	
	ATC		39.75	
	GP - standard terminals	29.88	33.56	42.76
	GP - additional terminals	26.92	31.23	36.90
machine eligibility precedence constraints	GA	27.61	28.93	30.86
	EDD		44.00	
	MS		47.42	
	MON		41.67	
	COVERT		44.23	
	ATC		42.58	
	GP - standard terminals	35.77	39.51	56.31
	GP - additional terminals	34.81	39.93	54.32
setup times precedence constraints	GA	18.19	19.83	22.70
	EDD		31.08	
	MS		33.61	
	MON		30.29	
	COVERT		30.35	
	ATC		27.66	
	GP - standard terminals	25.06	28.37	30.19
	GP - additional terminals	25.88	29.54	32.05

Table 9: Results obtained for combinations of three and four additional scheduling constraints

		min	med	max
setup times, machine eligibility, machine unavailability	GA	22.31	22.71	23.72
	EDD		65.07	
	MS		68.49	
	MON		57.42	
	COVERT		60.24	
	ATC		37.86	
	GP - standard terminals	37.98	42.33	47.89
	GP - additional terminals	35.98	41.97	52.49
setup times, machine unavailability, precedence constraints	GA	22.59	23.66	26.45
	EDD		58.63	
	MS		68.68	
	MON		56.21	
	COVERT		58.04	
	ATC		55.95	
	GP - standard terminals	39.80	43.81	52.12
	GP - additional terminals	34.40	42.84	47.77
setup times, machine eligibility, precedence constraints	GA	32.67	34.59	36.86
	EDD		59.92	
	MS		63.67	
	MON		54.52	
	COVERT		62.01	
	ATC		56.99	
	GP - standard terminals	46.38	51.47	58.59
	GP - additional terminals	46.73	51.59	58.60
machine eligibility, machine unavailability, precedence constraints	GA	36.30	37.90	39.75
	EDD		102.0	
	MS		95.33	
	MON		89.19	
	COVERT		86.90	
	ATC		83.09	
	GP - standard terminals	59.47	67.50	73.25
	GP - additional terminals	58.29	63.79	68.35
setup times, machine eligibility, machine unavailability, precedence constraints	GA	42.33	44.06	46.63
	EDD		123.1	
	MS		118.32	
	MON		111.8	
	COVERT		108.7	
	ATC		108.3	
	GP - standard terminals	76.61	80.72	106.6
	GP - additional terminals	70.42	81.99	91.19

designed rules to perform good decisions. The difference is especially evident for the case when all four constraints are used simultaneously. In this case the automatically designed DRs perform around 34% better than the ATC rule, which achieves the best performance among the manually designed rules. Such results demonstrate the superiority of automatically designed DRs over manually designed ones.

The results also demonstrate that for most constraint combinations there is no significant difference between DRs generated with and without using additional terminal nodes. The median values in both scenarios are usually quite similar, but neither approach di consistently outperform the other. However, one thing which is evident from the results is that by using the additional terminal nodes GP does obtain better minimum values in more situations. Therefore the terminal nodes do provide useful information to the DRs, however, it seems that the problem lies more in the evolution process that has difficulties to locate such rules. Once again one has to take into account that these results could have probably been further improved, since the node combinations were not optimised in this case, but were just selected based on the results obtained when considering the individual constraints. Also, as more constraints were considered the number of terminals that were at the disposal to GP grew. Therefore, the search space grow and it becomes more difficult for GP to generate good DRs.

7. Discussion

Based on the results that were outlined in the last section, several interesting conclusions can be drawn. First of all, it is evident that not all constraints are equally difficult for DRs to handle. This can be seen when comparing the results of the DRs with the results obtained by the GA. In some cases the differences between those methods are less prominent. This usually happens when wither the machine eligibility or precedence constraints are considered. The reason why the performance difference is smaller is due to the fact that these two constraints actively restrict the decisions that DRs can make at each moment. This means that at each decision point DRs will have a smaller set of jobs to choose from, or less machines on which a job can execute. Therefore, it is easier for DRs to reach a better decision. On the other hand, the remaining constraints do not impose a restriction on the decisions that can be performed in each moment. However, because of the additional constraints the DRs need to take into account more information. Therefore, the number of decisions from which the rules need to select the best one remains the same; however, the decision depends on more parameters, which makes selecting the right job and machine more difficult than in the case when no constraints are used.

The real benefit of the automatically designed DRs becomes evident when these rules are compared to manually designed ones. Out of the 15 different constraint combinations that were tested, automatically designed DRs obtained a better median value than the best manually designed rule in 12 experiments. This shows that the average performance of the generated rules is better in most cases, and that GP can construct DRs which are generally better than their manually designed counterparts. For every single constraint combination GP obtained a DR that performs better than any of the automatically designed rules. Such good results are almost equally obtained regardless whether the DRs are designed with additional terminal nodes or not. This shows that manually designed rules are limited and not suitable for the different scheduling problem variants, regardless of the adaptations in the SGS. GP demonstrated that it can obtain DRs suitable for solving problems with specific constraints. This is due to the fact that it can construct the PF which is well adapted to the problem at hand. Thus, this method offers unparalleled flexibility. Although manually designed rules could have further been manually adapted for additional constraints, such adaptations are difficult, time consuming, and require a lot of expert knowledge. In most cases they are not trivial, and are usually performed in a trial and error manner. Additionally, in the case of the ATC rule, the results show that the adapted version of this rule for setup time constraints actually performed worse than other manually designed DRs which did not take setup times into account. This means that the performance of the ATC rule is quite sensitive on the choice of the control parameters. Even though the test and train sets were similar, the rule performed quite poorly for the optimal value for the control parameter that was obtained on the training set. Although this control parameter should serve to provide a greater flexibility for this rule, it proves to be difficult to select the right value for it. Automatically designing DRs with GP provides much more flexibility. First of all, they have demonstrated to be more versatile and less sensitive of the problems they it solve in comparison to manually designed DRs. Secondly, GP can quite easily obtain DRs which perform better than manually designed ones for almost all the tested constraint combinations. Not only that, for several experiment these improvements were quite large.

The benefit of using additional terminal nodes designed for the different constraints is more difficult to asses. In most cases, the differences with and without them are relatively small. However, using those nodes can lead to significantly better results in some cases, and in most cases it can result in GP obtaining better minimum values. When a single constraint is considered, the terminal nodes have shown to be more useful when used for constraints that do not restrict the number of decisions that the DR can make in each step, like the setup times and machine availability constraints. The reason why in these cases such terminal nodes are important is due to the fact that the DR has to consider more information based on which it needs to perform the decision. Therefore, including different information about such constraints into the PF gives the DR a better overview of the problem and helps it to perform better decisions during scheduling. For the other two constraints the SGS restricts the decisions that can be made by the DR at certain moments. Therefore, the DR already has a smaller set of candidates to choose from, and additional terminal nodes might not be then useful in such scenarios. Regarding the individual constraints, automatically generated DRs performed quite well for three of them. For the precedence constraint GP did exhibit some difficulties in constructing good DRs.

The reason why this criteria is difficult for GP is due to the fact that it can be hard for it capture all the peculiarities of this constraint, so that the DR can perform good decisions. Unfortunately, even testing a large number of nodes for this constraint was not enough to improve its performance. Therefore, this constraint needs to be either handled in a different way, or more informative terminals would need to be designed.

When several constraints are considered simultaneously, there are no exact rules by which it could be determined for which constraints it makes sense to use additional terminals. In most cases the variant which uses and the one which does not use additional terminals performed quite similarly, without any significant difference in their results. It must be taken into account that the set of terminal nodes that was used for these constraint combinations used was not optimised for each combination individually. Rather, all the nodes for which the best results were obtained for the individual constraints were used, which can lead to large sets of terminal nodes when several constraints are considered simultaneously. Including more nodes increases the search space of GP drastically and makes it difficult for GP to locate good DRs. Therefore, by a better selection of nodes, or the design of novel terminal nodes for constraint combinations, the performance of the automatically designed DRs could be further improved. However, even in this case when the set of nodes was not optimised, for most constraints the performance of automatically designed DRs was better than that of their manually designed counterparts.

All the results demonstrate that for scheduling problems with several constraints GP can generate DRs which perform better than existing manually designed DRs. Even for such more complicated problems GP is expressive enough to design high quality DRs. As these automatically designed DRs performed better than several manually designed ones, designing DRs with GP represents a better alternative than adapting existing DRs. Since real world problems usually contain many different constraints, the results demonstrate that even for these cases GP could generate good quality DRs. Naturally, to adapt this approach for such situations one would first have to ensure that a corresponding training set exists which can be used by GP to learn the characteristics of the problems. Additionally, the SGS of the DRs needs to be adapted to ensure that only feasible solutions are constructed. These two conditions are enough to generate DRs for practically any new constraint that could appear in real world problems. Additionally, creating terminal nodes which provide information about the considered constraints is also suggested, as it should allow GP to obtain even better DRs for certain constraint types.

8. Conclusion

This study deals with the problem of adapting automatically designed DRs for solving scheduling problems which include different constraints. To tackle this problem the SGS that is used by automatically designed DRs was adapted to take into consideration all the constraints and to ensure that only feasible schedules will be created. Additionally, to give the DRs a better overview on the problem and further improve their performance, a set of additional terminal nodes that include information about each of the constraints was defined. The approach was used to generate DRs for four scheduling constraints individually and in different combinations. The performance was validated in comparison with five selected manually designed DRs, which were also adapted for problems with constraints.

The obtained results proved that automatically designed DRs are more versatile and have a better performance than their manually designed counterparts. Automatically designed DRs achieved better results for 12 out of the 15 considered cases, with improvements of even 30% for certain constraint combinations. Although manually designed DRs can also be adapted to consider different constraints, their limited performance soon becomes evident. GP offers the flexibility of designing rules which encompass the peculiarities of the problem they need to solve. As such, GP can easily find rules with good performance, which are also general enough that they can be applied to new problems and still retain their good performance. This approach also requires less expert knowledge, as it takes over the role of creating a meaningful PF from of the user. When all things are considered, GP demonstrated that it is expressive enough to generate DRs for complicated scheduling problems which include several constraints. As real world scheduling problems usually include many constraints, this observation is important as it demonstrates that high quality DRs could be automatically generated even for such situations as well.

Since the topic considered in this paper is quite extensive, there are many parts in which it could be extended in the future. One possible research direction would be to improve manually designed rules for the considered constraints to obtain a better baseline to which automatically generated DRs could be compared. Another part in which improvements could be made are in the design of terminal nodes. For each criteria it would be interesting to improve or add new terminal nodes. This is especially true for combinations of several criteria, for which it could make sense to design new terminal nodes which combine information regarding two or more constraints. For this part, the DRs which were obtained in this research could be used to analyse which terminals were most used and how the different terminals interacted. Finally, another constraint which was not included in this research, namely batch processing, is also planned to be considered in the future.

References

M. L. Pinedo, Scheduling: Theory, algorithms, and systems: Fourth edition, volume 9781461423614, Springer US, Boston, MA, 2012. URL: `http://link.springer.com/10.1007/978-1-4614-2361-4`. doi:`10.1007/978-1-4614-2361-4`. arXiv:`arXiv:1011.1669v3`.

M. Kofler, S. Wagner, A. Beham, G. Kronberger, M. Affenzeller, Priority Rule Generation with a Genetic Algorithm to Minimize Sequence Dependent Setup

Costs, in: R. Moreno-Díaz, F. Pichler, A. Quesada-Arencibia (Eds.), Computer Aided Systems Theory - EUROCAST 2009: 12th International Conference, Las Palmas de Gran Canaria, Spain, February 15-20, 2009, Revised Selected Papers, Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, pp. 817–824. URL: `http://link.springer.com/10.1007/978-3-642-04772-5_105`. doi:`10.1007/978-3-642-04772-5\_105`.

D. Ouelhadj, S. Petrovic, A survey of dynamic scheduling in manufacturing systems, Journal of Scheduling 12 (2009) 417–431.

V. Cheng, L. Crawford, P. Menon, Air traffic control using genetic search techniques, in: Proceedings of the 1999 IEEE International Conference on Control Applications (Cat. No.99CH36328), volume 1, IEEE, 1999, pp. 249–254. URL: `http://ieeexplore.ieee.org/document/806209/`. doi:`10.1109/CCA.1999.806209`.

J. V. Hansen, Genetic search methods in air traffic control, Computers & Operations Research 31 (2004) 445–459.

F. Corman, E. Quaglietta, Closing the loop in real-time railway control: Framework design and impacts on operations, Transportation Research Part C: Emerging Technologies 54 (2015) 15 – 39.

E. Burke, P. De Causmaecker, G. Vanden Berghe, H. Van Landeghem, The state of the art of nurse rostering, J. Scheduling 7 (2004) 441–499.

S. Petrovic, E. Castro, A genetic algorithm for radiotherapy pre-treatment scheduling, in: C. Di Chio, A. Brabazon, G. A. Di Caro, R. Drechsler, M. Farooq, J. Grahl, G. Greenfield, C. Prins, J. Romero, G. Squillero, E. Tarantino, A. G. B. Tettamanzi, N. Urquhart, A. Ş. Uyar (Eds.), Applications of Evolutionary Computation: EvoApplications 2011: EvoCOMNET, EvoFIN, EvoHOT, EvoMUSART, EvoSTIM, and EvoTRANSLOG, Torino, Italy, April 27-29, 2011, Proceedings, Part II, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, pp. 454–463.

R. Lewis, B. Paechter, O. Rossi-Doria, Metaheuristics for University Course Timetabling, Springer Berlin Heidelberg, Berlin, Heidelberg, 2007, pp. 237–272. URL: `https://doi.org/10.1007/978-3-540-48584-1_9`. doi:`10.1007/978-3-540-48584-1_9`.

E. Hart, P. Ross, D. Corne, Evolutionary Scheduling: A Review, Genetic Programming and Evolvable Machines 6 (2005) 191–220.

I. Vlašić, M. Ðurasević, D. Jakobović, Improving genetic algorithm performance by population initialisation with dispatching rules, Computers and Industrial Engineering 137 (2019) 106030.

I. Vlašić, M. Ðurasević, D. Jakobović, A comparative study of solution representations for the unrelated machines environment, Computers & Operations Research 123 (2020) 105005.

D.-W. Kim, K.-H. Kim, W. Jang, F. F. Chen, Unrelated parallel machine scheduling with setup times using simulated annealing, Robotics and Computer-Integrated Manufacturing 18 (2002) 223 – 231. 11th International Conference on Flexible Automation and Intelligent Manufacturing.

J.-H. Lee, J.-M. Yu, D.-H. Lee, A tabu search algorithm for unrelated parallel machine scheduling with sequence- and machine-dependent setups: minimizing total tardiness, The International Journal of Advanced Manufacturing Technology 69 (2013) 2081–2089.

J. Behnamian, M. Zandieh, S. Fatemi Ghomi, Parallel-machine scheduling problems with sequence-dependent setup times using an ACO, SA and VNS hybrid algorithm, Expert Systems with Applications 36 (2009) 9637–9644.

M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, R. F. Freund, Dynamic Mapping of a Class of Independent Tasks onto Heterogeneous Computing Systems, Journal of Parallel and Distributed Computing 59 (1999) 107–131.

T. D. Braun, H. J. Siegel, N. Beck, L. L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen, R. F. Freund, A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems, Journal of Parallel and Distributed Computing 61 (2001) 810–837.

M. Đurasević, D. Jakobović, A survey of dispatching rules for the dynamic unrelated machines environment, Expert Systems with Applications 113 (2018) 555 – 569.

J. Branke, S. Nguyen, C. W. Pickardt, M. Zhang, Automated Design of Production Scheduling Heuristics: A Review, IEEE Transactions on Evolutionary Computation 20 (2016) 110–124.

S. Nguyen, Y. Mei, M. Zhang, Genetic programming for production scheduling: a survey with a unified framework, Complex & Intelligent Systems 3 (2017) 41–66.

J. Koza, Genetically breeding populations of computer programs to solve problems in artificial intelligence, in: [1990] Proceedings of the 2nd International IEEE Conference on Tools for Artificial Intelligence, IEEE Comput. Soc. Press, 1990, pp. 819–827. URL: `http://ieeexplore.ieee.org/document/130444/`. doi:`10.1109/TAI.1990.130444`.

R. Poli, W. B. Langdon, N. F. McPhee, A field guide to genetic programming, Published via `http://lulu.com` and freely available at `http://www.gp-field-guide.org.uk`, 2008. URL: `http://www.gp-field-guide.org.uk`, (With contributions by J. R. Koza).

E. K. Burke, M. R. Hyde, G. Kendall, J. Woodward, Automatic heuristic generation with genetic programming, in: Proceedings of the 9th annual conference on Genetic and evolutionary computation - GECCO '07, ACM Press, New York, New York, USA, 2007, p. 1559. URL: http://portal.acm.org/citation.cfm?doid=1276958.1277273. doi:10.1145/1276958.1277273.

E. K. Burke, M. R. Hyde, G. Kendall, G. Ochoa, E. Ozcan, J. R. Woodward, Exploring Hyper-heuristic Methodologies with Genetic Programming, Computational Intelligence 1 (2009) 177–201.

E. K. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, R. Qu, Hyper-heuristics: a survey of the state of the art, Journal of the Operational Research Society 64 (2013) 1695–1724.

E. K. Burke, M. R. Hyde, G. Kendall, J. Woodward, Automating the Packing Heuristic Design Process with Genetic Programming, Evolutionary Computation 20 (2012) 63–89.

N. Pillay, R. Qu, Hyper-Heuristics: Theory and Applications, Springer International Publishing, 2018. URL: http://dx.doi.org/10.1007/978-3-319-96514-7. doi:10.1007/978-3-319-96514-7.

Y. Mei, M. Zhang, Genetic programming hyper-heuristic for multi-vehicle uncertain capacitated arc routing problem, in: Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO '18, Association for Computing Machinery, New York, NY, USA, 2018, p. 141–142. URL: https://doi.org/10.1145/3205651.3205661. doi:10.1145/3205651.3205661.

Y. Liu, Y. Mei, M. Zhang, Z. Zhang, A predictive-reactive approach with genetic programming and cooperative coevolution for the uncertain capacitated arc routing problem, Evolutionary Computation (2019) 1–28. PMID: 31012736.

C. Dimopoulos, A. Zalzala, A genetic programming heuristic for the one-machine total tardiness problem, in: Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406), IEEE, 1999, pp. 2207–2214. URL: http://ieeexplore.ieee.org/document/785549/. doi:10.1109/CEC.1999.785549.

C. Dimopoulos, A. Zalzala, Investigating the use of genetic programming for a classic one-machine scheduling problem, Advances in Engineering Software 32 (2001) 489–498.

K. Miyashita, Job-shop scheduling with genetic programming, in: Proceedings of the 2Nd Annual Conference on Genetic and Evolutionary Computation, GECCO'00, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2000, pp. 505–512. URL: http://dl.acm.org/citation.cfm?id=2933718.2933809.

J. C. Tay, N. B. Ho, Designing Dispatching Rules to Minimize Total Tardiness, in: K. P. Dahal, K. C. Tan, P. I. Cowling (Eds.), Evolutionary Scheduling, Springer Berlin Heidelberg, Berlin, Heidelberg, 2007, pp. 101–124. URL: http://link.springer.com/10.1007/978-3-540-48584-1_4. doi:10.1007/978-3-540-48584-1_4.

D. Jakobović, L. Jelenković, L. Budin, Genetic Programming Heuristics for Multiple Machine Scheduling, in: Genetic Programming, Springer Berlin Heidelberg, Berlin, Heidelberg, 2007, pp. 321–330. URL: `http://link.springer.com/10.1007/978-3-540-71605-1_30`. doi:`10.1007/978-3-540-71605-1\_30`.

M. Đurasević, D. Jakobović, K. Knežević, Adaptive scheduling on unrelated machines with genetic programming, Applied Soft Computing 48 (2016) 419–430.

M. Đurasević, D. Jakobović, Comparison of schedule generation schemes for designing dispatching rules with genetic programming in the unrelated machines environment, Applied Soft Computing 96 (2020) 106637.

S. Nguyen, M. Zhang, M. Johnston, K. C. Tan, Learning Reusable Initial Solutions for Multi-objective Order Acceptance and Scheduling Problems with Genetic Programming, in: K. Krawiec, A. Moraglio, T. Hu, A. Ş. Etaner-Uyar, B. Hu (Eds.), Genetic Programming: 16th European Conference, EuroGP 2013, Vienna, Austria, April 3-5, 2013. Proceedings, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 157–168. URL: `http://link.springer.com/10.1007/978-3-642-37207-0_14`. doi:`10.1007/978-3-642-37207-0\_14`.

S. Nguyen, M. Zhang, M. Johnston, A sequential genetic programming method to learn forward construction heuristics for order acceptance and scheduling, in: 2014 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2014, pp. 1824–1831. URL: `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6900347`. doi:`10.1109/CEC.2014.6900347`.

S. Chand, Q. Huynh, H. Singh, T. Ray, M. Wagner, On the use of genetic programming to evolve priority rules for resource constrained project scheduling problems, Information Sciences 432 (2018) 146 – 163.

M. Đumić, D. Šišejković, R. Čorić, D. Jakobović, Evolving priority rules for resource constrained project scheduling problem with genetic programming, Future Generation Computer Systems 86 (2018) 211 – 221.

S. Nguyen, M. Zhang, M. Johnston, K. C. Tan, Dynamic multi-objective job shop scheduling: A genetic programming approach, in: A. S. Uyar, E. Ozcan, N. Urquhart (Eds.), Automated Scheduling and Planning: From Theory to Practice, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 251–282.

S. Nguyen, M. Zhang, K. C. Tan, Enhancing genetic programming based hyper-heuristics for dynamic multi-objective job shop scheduling problems, in: 2015 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2015, pp. 2781–2788. URL: `http://ieeexplore.ieee.org/document/7257234/`. doi:`10.1109/CEC.2015.7257234`.

D. Karunakaran, G. Chen, M. Zhang, Parallel Multi-objective Job Shop Scheduling Using Genetic Programming, in: T. Ray, R. Sarker, X. Li (Eds.), Artificial Life and Computational Intelligence: Second Australasian Conference, ACALCI 2016, Canberra, ACT, Australia, February 2-5, 2016, Proceedings, Springer International Publishing, 2016, pp. 234–245. URL: `http://link.springer.com/10.1007/978-3-319-28270-1_20`. doi:`10.1007/978-3-319-28270-1\_20`.

M. Durasević, D. Jakobović, Evolving dispatching rules for optimising many-objective criteria in the unrelated machines environment, Genetic Programming and Evolvable Machines 19 (2018) 9–51.

D. Jakobović, L. Budin, Dynamic scheduling with genetic programming, in: P. Collet, M. Tomassini, M. Ebner, S. Gustafson, A. Ekárt (Eds.), Genetic Programming: 9th European Conference, EuroGP 2006, Budapest, Hungary, April 10-12, 2006. Proceedings, Springer Berlin Heidelberg, Berlin, Heidelberg, 2006, pp. 73–84.

J. Park, S. Nguyen, M. Zhang, M. Johnston, Evolving ensembles of dispatching rules using genetic programming for job shop scheduling, in: P. Machado, M. I. Heywood, J. McDermott, M. Castelli, P. García-Sánchez, P. Burelli, S. Risi, K. Sim (Eds.), Genetic Programming: 18th European Conference, EuroGP 2015, Copenhagen, Denmark, April 8-10, 2015, Proceedings, Springer International Publishing, Cham, 2015, pp. 92–104. URL: `https://doi.org/10.1007/978-3-319-16501-1_8`. doi:`10.1007/978-3-319-16501-1_8`.

E. Hart, K. Sim, A Hyper-Heuristic Ensemble Method for Static Job-Shop Scheduling, Evolutionary Computation 24 (2016) 609–635.

J. Park, Y. Mei, S. Nguyen, G. Chen, M. Zhang, An investigation of ensemble combination schemes for genetic programming based hyper-heuristic approaches to dynamic job shop scheduling, Applied Soft Computing 63 (2018) 72 – 86.

M. Durasević, D. Jakobović, Comparison of ensemble learning methods for creating ensembles of dispatching rules for the unrelated machines environment, Genetic Programming and Evolvable Machines (2017).

M. Đurasević, D. Jakobović, Creating dispatching rules by simple ensemble combination, Journal of Heuristics 25 (2019) 959–1013.

F. Zhang, Y. Mei, M. Zhang, Genetic programming with multi-tree representation for dynamic flexible job shop scheduling, in: T. Mitrovic, B. Xue, X. Li (Eds.), AI 2018: Advances in Artificial Intelligence, Springer International Publishing, Cham, 2018, pp. 472–484.

J. Branke, T. Hildebrandt, B. Scholz-Reiter, Hyper-heuristic Evolution of Dispatching Rules: A Comparison of Rule Representations, Evolutionary Computation 23 (2015) 249–277.

S. Nguyen, M. Zhang, M. Johnston, K. C. Tan, A Computational Study of Representations in Genetic Programming to Evolve Dispatching Rules for the Job Shop Scheduling Problem, IEEE Transactions on Evolutionary Computation 17 (2013) 621–639.

S. Nguyen, Y. Mei, B. Xue, M. Zhang, A hybrid genetic programming algorithm for automated design of dispatching rules, Evolutionary Computation 27 (2019) 467–496. PMID: 29863420.

L. Nie, X. Shao, L. Gao, W. Li, Evolving scheduling rules with gene expression programming for dynamic single-machine scheduling problems, The International Journal of Advanced Manufacturing Technology 50 (2010) 729–747.

L. Nie, L. Gao, P. Li, L. Zhang, Application of gene expression programming on dynamic job shop scheduling problem, in: Proceedings of the 2011 15th International Conference on Computer Supported Cooperative Work in Design (CSCWD), IEEE, 2011, pp. 291–295. URL: `http://ieeexplore.ieee.org/document/5960088/`. doi:`10.1109/CSCWD.2011.5960088`.

L. Nie, L. Gao, P. Li, X. Li, A GEP-based reactive scheduling policies constructing approach for dynamic flexible job shop scheduling problem with job release dates, Journal of Intelligent Manufacturing 24 (2013) 763–774.

M. Đurasević, D. Jakobović, Automatic design of dispatching rules for static scheduling conditions, Neural Computing and Applications (2020).

E. Pitzer, A. Beham, M. Affenzeller, H. Heiss, M. Vorderwinkler, Production fine planning using a solution archive of priority rules, in: 3rd IEEE International Symposium on Logistics and Industrial Informatics, IEEE, 2011, pp. 111–116. URL: `http://ieeexplore.ieee.org/document/6031130/`. doi:`10.1109/LINDI.2011.6031130`.

C. W. Pickardt, T. Hildebrandt, J. Branke, J. Heger, B. Scholz-Reiter, Evolutionary generation of dispatching rule sets for complex dynamic scheduling problems, International Journal of Production Economics 145 (2013) 67–77.

D. Jakobović, K. Marasović, Evolving priority scheduling heuristics with genetic programming, Applied Soft Computing 12 (2012) 2781–2789.

J. Park, S. Nguyen, M. Zhang, M. Johnston, Genetic programming for order acceptance and scheduling, in: 2013 IEEE Congress on Evolutionary Computation, IEEE, 2013, pp. 1005–1012. URL: `http://ieeexplore.ieee.org/document/6557677/`. doi:`10.1109/CEC.2013.6557677`.

A. Beham, S. Winkler, S. Wagner, M. Affenzeller, A genetic programming approach to solve scheduling problems with parallel simulation, in: 2008 IEEE International Symposium on Parallel and Distributed Processing, IEEE, 2008, pp. 1–5. URL: `http://ieeexplore.ieee.org/document/4536362/`. doi:`10.1109/IPDPS.2008.4536362`.

J. C. Tay, N. B. Ho, Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems, Computers & Industrial Engineering 54 (2008) 453–473.

Wen-Jun Yin, Min Liu, Cheng Wu, Learning single-machine scheduling heuristics subject to machine breakdowns with genetic programming, in: The 2003 Congress on Evolutionary Computation, 2003. CEC '03., volume 2, IEEE, 2003, pp. 1050–1055. URL: `http://ieeexplore.ieee.org/document/1299784/`. doi:`10.1109/CEC.2003.1299784`.

A. P. J. Vepsalainen, T. E. Morton, Priority Rules for Job Shops with Weighted Tardiness Costs, Management Science 33 (1987) 1035–1047.

Y. H. Lee, K. Bhaskaran, M. Pinedo, A heuristic to minimize the total weighted tardiness with sequence-dependent setups, IIE Transactions 29 (1997) 45–52.

M. Pfund, J. W. Fowler, A. Gadkari, Y. Chen, Scheduling jobs on parallel machines with setup times and ready times, Computers & Industrial Engineering 54 (2008) 764–782.